TELEOPERATED POSITION CONTROL OF A PUMA ROBOT

Edmund Austin and Chung P. Fong, Ph.D.

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Abstract

A laboratory distributed computer control teleoperator system is developed to support NASA's future space telerobotic operation. This teleoperator system uses a universal force-reflecting hand controller in the local site as the operator's input device. In the remote site, a PUMA controller receives the Cartesian position commands and implements PID control laws to position the PUMA robot. The local site uses two microprocessors while the remote site uses three. The processors communicate with each other through shared memory. The PUMA robot controller was interfaced through custom made electronics to bypass VAL.

In this paper, the development status of this teleoperator system is reported. The execution time of each processor is analyzed, and the overall system throughput rate is reported. Methods to improve the efficiency and performance are discussed.

1.0 INTRODUCTION

It is hoped that in the future mankind will inhabit space on a permanent basis. Whether it be military crews operating space based offensive/defensive facilities or civilians living in space colonies, the habitation of space will require large amounts of construction and repair in space. For such tasks, it is preferable not to use astronauts since radiation hazards would limit the amount of time one person could be allotted EVAs (Extra Vehicular Activity). Also, there are some tasks where safety considerations would preclude using a man at all. In these situations it would be preferable to use some type of robotic device to achieve one's objectives.

Basically there are two types of robotic devices to use: autonomous and teleoperated. Autonomous systems require no human assistance to accomplish their task. After being informed of the task to perform, the autonomous system executes the task either from some preset repertoire of tasks or uses some type of artificial intelligence to determine how to tackle the problem. This requires that you have one or more of the following: 1) a very large set of pre-defined tasks to cover any and all eventualities, 2) a very good model of the environment, 3) a very powerful artificial intelligence computing capability, or 4) a good method of incorporating sensor data. This is a partial list of autonomous system requirements and still they may be difficult to meet.

An alternative is a teleoperated system. A teleoperated system is basically a robotic device that is remotely controlled by a human operator. So we will depend upon a human mind to determine how to tackle a task. Still, for a human operator to properly direct a robot he will need sensors not only to provide him with a view of what the robot is doing, but also a feel for the forces the robot is both exerting and experiencing.

In order to study some of these teleoperation issues we have constructed a teleoperation system consisting of a force reflecting hand controller, a Unimation 560 robot, and five National Semiconductor microprocessors. The microprocessors are needed to perform kinematic transformation and data communication since the FRHC (Force Reflecting Hand Controller) and robot are kinematically dissimilar and physically separated. With this system we will attempt to determine what defines "good" teleoperation and how to improve it.

2.0 SYSTEM ARCHITECTURE

The computing hardware of our system consists of five National Semiconductor 32016 CPU development boards with N.S. 16081 floating point unit and an 10 megahertz clock (except the FRHC control CPU which has a 6 megahertz clock), two BLC-519 I/O boards, two 128 kilobyte RAM boards, and some ancillary electronics to interface with the FRHC (Force Reflecting Hand Controller) and the Unimation PUMA 560 robot.

All of the computing hardware is contained in two Multibus Chassis. See Figure 1. More specifically, the various components perform the following tasks:

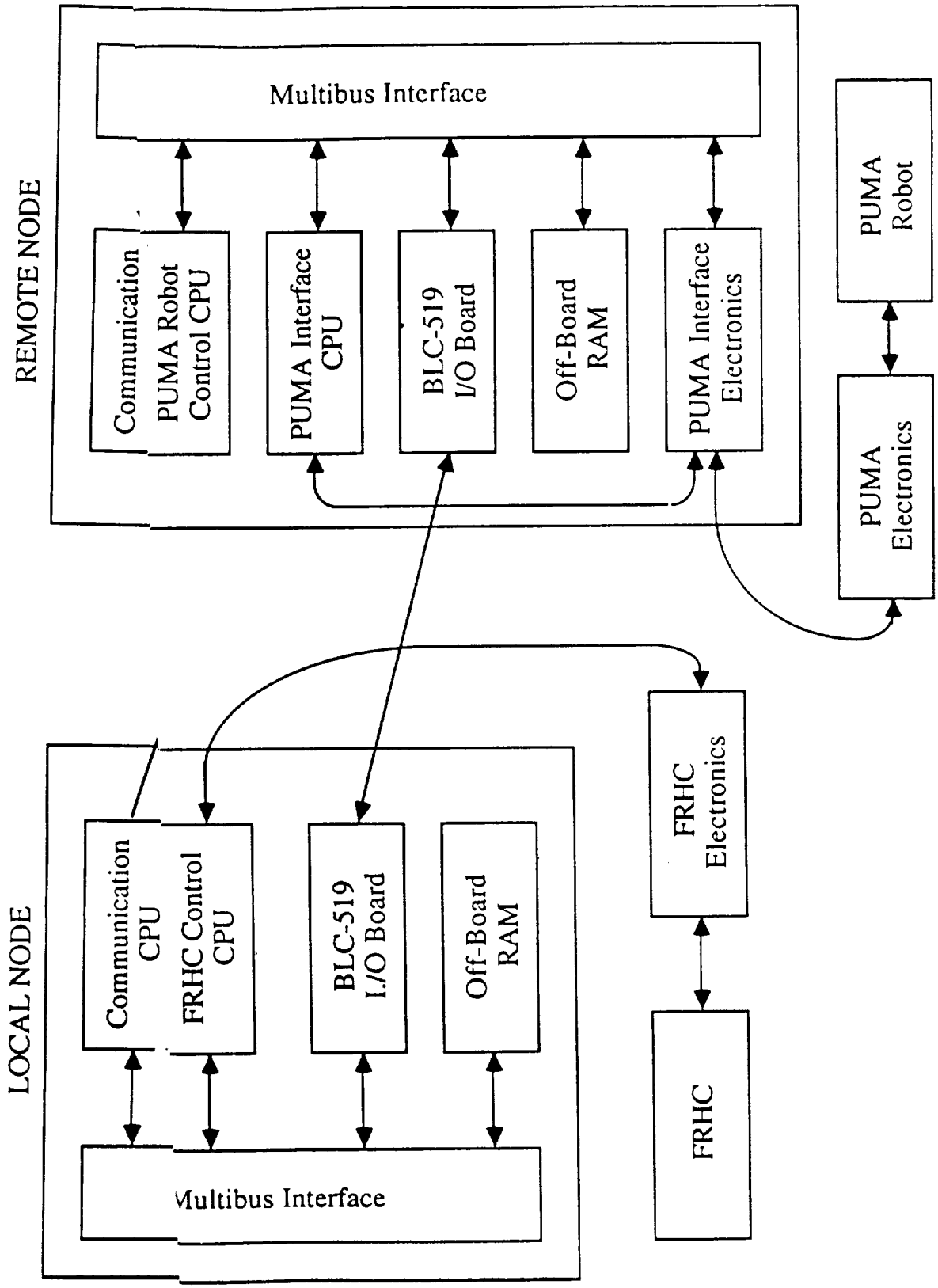

Figure 1. System Architecture

1) Two communications CPUs, one located in each chassis (node), that decide what data to send, retrieve that data from off board RAM, assemble that data into a buffer and send it over a parallel link. Conversely, when a communication CPU is receiving, it determines what data it is receiving, and places that data in the appropriate locations in off board RAM. The local communication CPU also contains the menu through which system parameters can be altered.

2) One FRHC control CPU that interprets the encoder values of the FRHC and converts them into joint angles, from these joint angles the Cartesian position and orientation of the end of the FRHC is determined (i.e. the FRHC T6 matrix), for position control. For rate control, the deviation of the FRHC from some neutral position is used to generate rate commands. This CPU also calculates the force feedback to backdrive the FRHC, whether we are in the rate or position mode.

3) Two BLC-519 I/O boards that have the 8 bit parallel I/O ports that the communications CPUs actually use to send and receive data between the remote and local nodes.

4) Two BLC-0128A 128 kilobyte off board RAM boards, that are used to hold all information that is shared between processors within a node and to hold all information shared between nodes. Each specific piece of stored information is held at a specific address known to all the CPUs.

5) The PUMA robot control CPU does the inverse kinematics and the forward kinematics of the robot, along with compensation for an end effector, and workspace transformations.

6) Actual interfacing with the robot is accomplished by the PUMA interface CPU. It sends joint angle commands to the robot, reads the current robot joint angles, calibrates the robot, and performs the setpoint interpolation.

7) The PUMA electronics provides the servo power to the robot and has six Motorola 6503 joint microprocessors that actually perform the low level robot servo control.

8) FRHC electronics allows us to read the potentiometers of the FRHC's six joints and to supply current to the motors attached to the six joints for force feedback.

9) The PUMA interface electronics facilitates direct communication with the six 6503 joint microprocessors.

PUMA Interface Electronics

Our interface design approach is to by-pass Unimation's LSI-11 resident VAL-II and to simulate Unimation's interface to the DRV11 in the PUMA arm controller by using two Intel 8255 Programmable Peripheral Interface(PPI) adapters. Port A, B, and C on the PPI are used for data input, output and handshake, respectively. Mode 1 is selected as the port mode. To read or write the data/commands from/to the joint processors, a request

signal hasent through handshake line. The data sampling rate and the position commandng rate is hence dependent on the high level processor cycle rate.

The l servo commands issued from high level processor are decomposed into digital semands acceptable by PUMA joint processors. The low level servo commands ardy the following four routines coded in "C":

a) read_int, command, data) - reads encoder/current value, depending on the commm the specified joint.

b) read_ommand, data) - reads an array of encoder/current values from all six joints

c) write_int, command, data) - write set points/motor currents, depending on the co, to the specified joint.

d) write_ommand, data) - write an array of set points/motor currents, to all six joints neously.

Embe the same interface software routine, those four routines interpret the commands form proper handshakes to read/write the data/digital servo commands to/from thA interface.

3.0 SYSTETROL MODES

Control Mration

The eoperation system is constituted of three control loops: the supervisory control loocal control loop and the remote control loop. The supervisory control loop includperator, the visual and audio feedback, and menu-driven commands. Through mection, the operator can switch the control mode, change the control parameters onitor the control status. The local control loop consists of the force reflecting hitroller and the local processors. The remote control loop is formed by the sensors anipulator and the remote processors. High level information, such as the Cartesiions/velocities and the control mode words, are exchanged between the local and the control loops.

In the control loop, the low level servo commands generated by the remote processors ad to the PUMA's joint processors through the interface board in the PUMA contnit. Since the PUMA joint processors control the joint motors either in "position or in "current mode," the servo commands generated by the processor are also bashe above two modes. In "position mode" the joint processor accepts encoder set and implements PID control. In "current mode" the motor torque commands acted by the joint processors. Currently, the PUMA interface processor takes advant the joint processor PID control capability and sends only encoder set points to theocessors. The direct joint motor control that sends current commands

to joint processors is not currently implemented because 1) the interrupt-driven interface, which facilitates the high bandwidth PD or PID servo control, is not installed; 2) the PUMA velocity needed for PD or PID control is also not available.

All of the following three major system control modes result in the position commands which servo the PUMA arm in its "position mode":

1) Position control mode. By reading the pots from the hand controller interface, the joint angles are computed. Forward kinematics of the hand controller is then performed to determine the end-point position and orientation in Cartesian coordinates, which is commonly referred to as the T6 matrix. Upon receiving the end-point positions of the hand controller through the parallel line, the remote processor performs the inverse kinematics of the PUMA to determine the desired joint positions. The desired joint set points in encoder values are then calculated and sent to the PUMA joint processor interface. The current PUMA joint position is also read by the remote processor. From this, robot forward kinematics are done to calculate the position and orientation of the robot's endpoint. This information is transmitted back to the local node. The position error between the hand controller position and PUMA position are then computed and used to back drive the hand controller.

2) Rate control mode. In rate control mode, the hand controller is utilized as a joystick-type input device, in which the spring effect is generated by the software. The joystick's displacement from its nominal position determines the rate. By integrating the rate, the local processor yields the aggregated positions which are then sent to the remote site. A small deadband was set around the nominal position of the hand controller to ensure no position output when the hand controller rests in the neighborhood of its nominal position. Since the feedback position is not processed in this mode, the throughput is slightly faster than in the position mode.

3) Mixed mode. In this mode, the translational axes operate in rate control mode while the rotational axes operate in position control mode. This mode would allow the operator to quickly move PUMA arm in the work space while maintaining the same orientation of the hand controller.

The above three control modes are actually commanding the PUMA arm's movement, and are therefore categorized as operation modes. The following are three other non-operation modes, in which the PUMA arm is "quiescent":

1) Start mode. This is the first mode entered upon turning on the power. In this mode, the initialization, health and status checking take place.

2) Index/menu mode. Since the work volume of the hand controller is much smaller than that of the PUMA arm, one of the means to augment the hand controller positioning capability is to use indexing. This mode allows the hand controller to move to a new position while the PUMA arm is frozen. The teleoperation resumes after a key on the keyboard or a button on the hand controller is hit. Another means of augmenting the

hand controller positioning capability is by using scaling factors. The movement of the PUMA arm can be amplified to a larger motion or confined to a smaller motion by the proper selection of scaling factors in the menu. The first one is for gross movement and the second one is for dexterous positioning. The scaling factors can be applied to both position control mode and rate control mode.

3) Quit mode. This mode is entered through menu selection. In this mode, procedures including robot servo power shut off, return of confirmation messages, etc., must be exercised prior to system shut- down.

Control Mode Transition

Since the robot is frozen during index mode, it is used as a natural gateway for mode transition. After startup, the system will default into the index/menu mode. From index/menu mode, the operator can transit into any of the following modes:

a) Quit mode - By menu selection.

b) Position mode - By depressing the index button on the handgrip of the hand controller.

c) Rate mode - By depressing the index button after selecting the rate mode from the menu.

d) Mixed mode - By depressing the index button after selecting the mixed mode from the menu. The mode transition diagram is shown in Fig. 2.

When exiting the index mode, the current PUMA position is stored and the translational bias between the new hand position and PUMA position is calculated. Provided that the indexing is also applicable to the rotation, as the operator desires, the rotational bias has to be calculated. The translational vector of the subsequent position commands are then added to the stored PUMA position to yield new PUMA position commands.

By using Denavit-Hartenberg notations, the translational bias is computed as follows:

$$({}^{a0}_{h0}T) = ({}^{a0}_{a6}T)\,({}^{a6}_{h6}T)\,({}^{ho}_{h6}T)^{-1} = \begin{bmatrix} \underline{R} & \underline{P} \\ \underline{0} & 1 \end{bmatrix} \tag{1}$$

where the $({}^{a0}_{h0}T)$ denotes the transformation from a0 (base of robot arm) to h0 (base of hand controller). Similarly, $({}^{a0}_{a6}T)$ is the current PUMA position (i.e., T6 matrix) w.r.t. its own base reference. $({}^{ho}_{h6}T)$ is the hand controller's new position w.r.t. its own base reference, and $({}^{a6}_{h6}T)$ is the coordinate transformation between PUMA arm's last joint and the hand controller's hand grip. The $\underline{R}$ matrix in $({}^{a0}_{h0}T)$ results in Eq. (1) being replaced by a 3x3 matrix, which is pre-determined to correct the coordinate disparity between hand controller base and robot base. The $\underline{P}$ vector in $({}^{a0}_{h0}T)$ represents the translational bias.

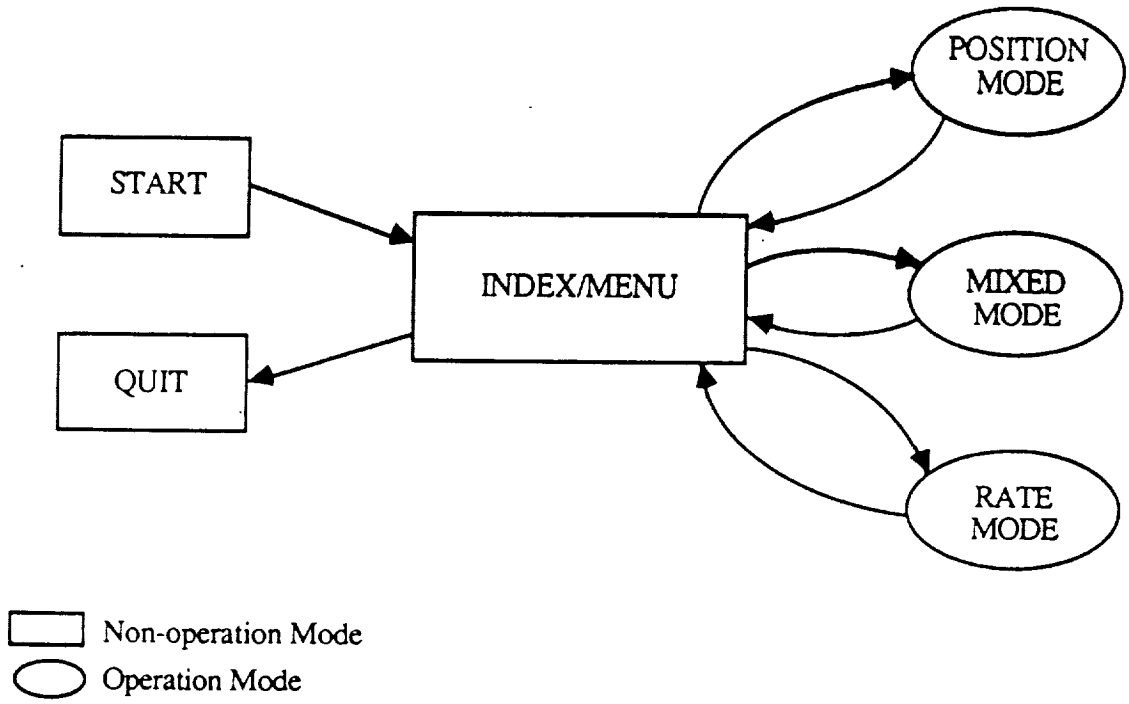

Figure 2. Mode Transition Diagram

The rotational bias is calculated as

$$({}^{a6}_{a6n}T) = ({}^{a0}_{a6}T)^{-1}({}^{a0}_{a6n}T) \tag{2}$$

where $({}^{a0}_{a6n}T)$ is determined by

$$({}^{a0}_{a6n}T) = ({}^{a0}_{h0}T)\,({}^{h0}_{h6}T)\,({}^{h6}_{a6}T) \tag{3}$$

and the $({}^{a0}_{h0}T)$ matrix is derived form Eq. (1). If the index on rotation is not desired, the $\underline{R}$ matrix in $({}^{a6}_{a6n}T)$ can be replaced by an identity matrix $\underline{I}$.

The subsequent position commands (T6) after indexing is then computed as follows:

$$({}^{a0}_{a6n}T) = ({}^{a0}_{h0}T)\,({}^{h0}_{h6}T)\,({}^{h6}_{a6}T)\,({}^{a6}_{a6n}T) \tag{4}$$

4.0 ROBOT KINEMATICS

In order to determine the position and orientation of the last link of the robot in Cartesian space, reference frames were assigned to each link of the robot, using Denavit-Hartenberg notation. See Figure 3. Six transformation matrices ${}^{i-1}_{i}T$ were then determined that relate the reference frame of link i to link $i-1$, which are a function of the joint angle of that link. Multiplying these six matrices, i.e.,

$${}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T\,{}^{5}_{6}T = {}^{0}_{6}T \tag{5}$$

yields a ${}^{0}_{6}T$ matrix, a 4x4 matrix that contains both the orientation and position of the reference frame of the last link with respect to some arbitrary reference frame.

$${}^{0}_{6}T = \begin{bmatrix} R_{11} & R_{12} & R_{13} & P_x \\ R_{21} & R_{22} & R_{23} & P_y \\ R_{31} & R_{32} & R_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \underline{R} & \underline{P} \\ \underline{0} & 1 \end{bmatrix} \tag{6}$$

where $\underline{R}$ is the orientation matrix and $\underline{P}$ is the position vector. This generation of a ${}^{0}_{6}T$ given the joint angles of the robot is referred to as the forward kinematics calculation.

There is one problem with this formulation thus far and that is that the reference frame of the last link is actually embedded within the robot wrist while what is desired is the position and orientation of the tip of an end effector. See Figure 4. Therefore we must

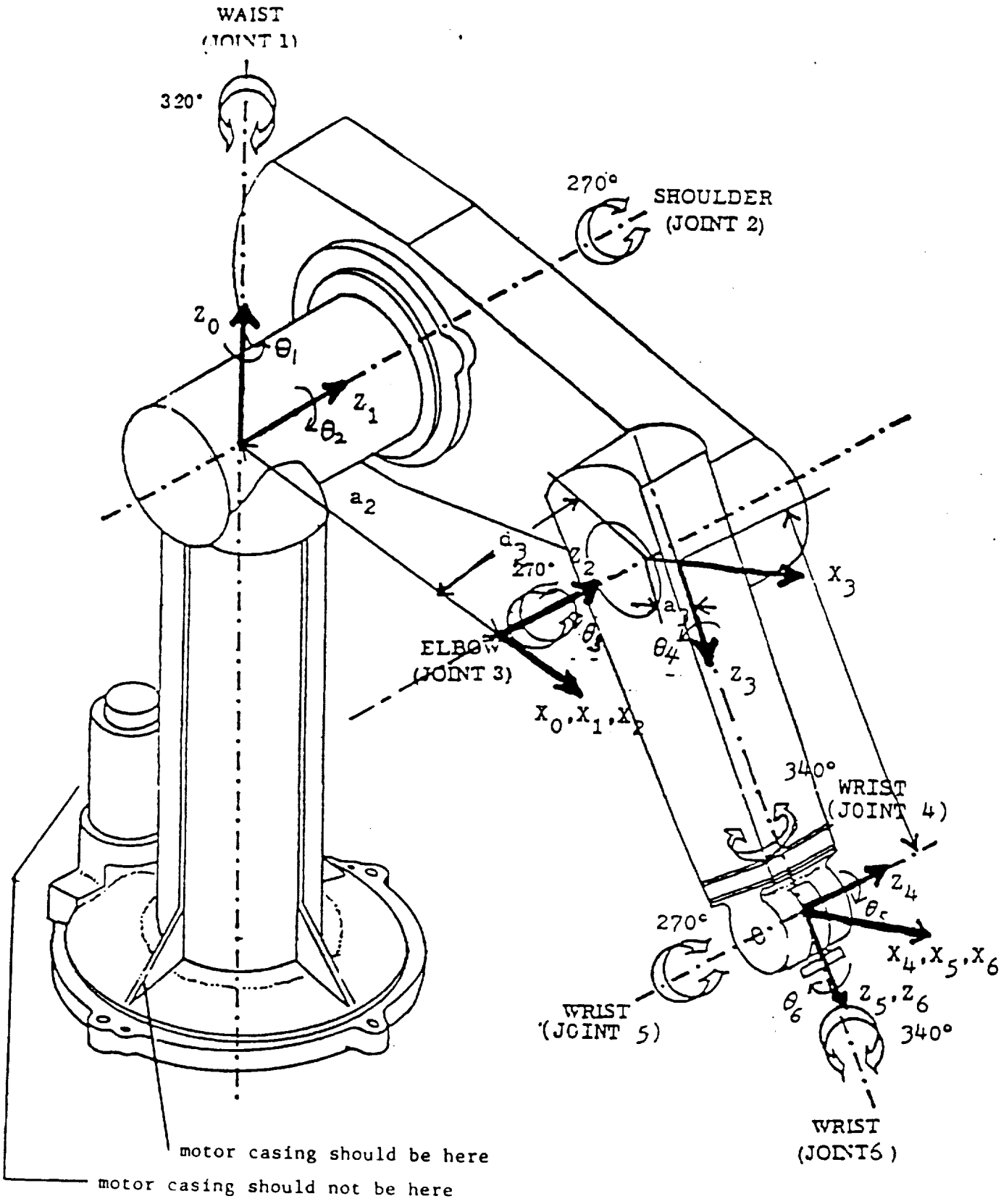

Figure 3. PUMA Robot Reference Frames

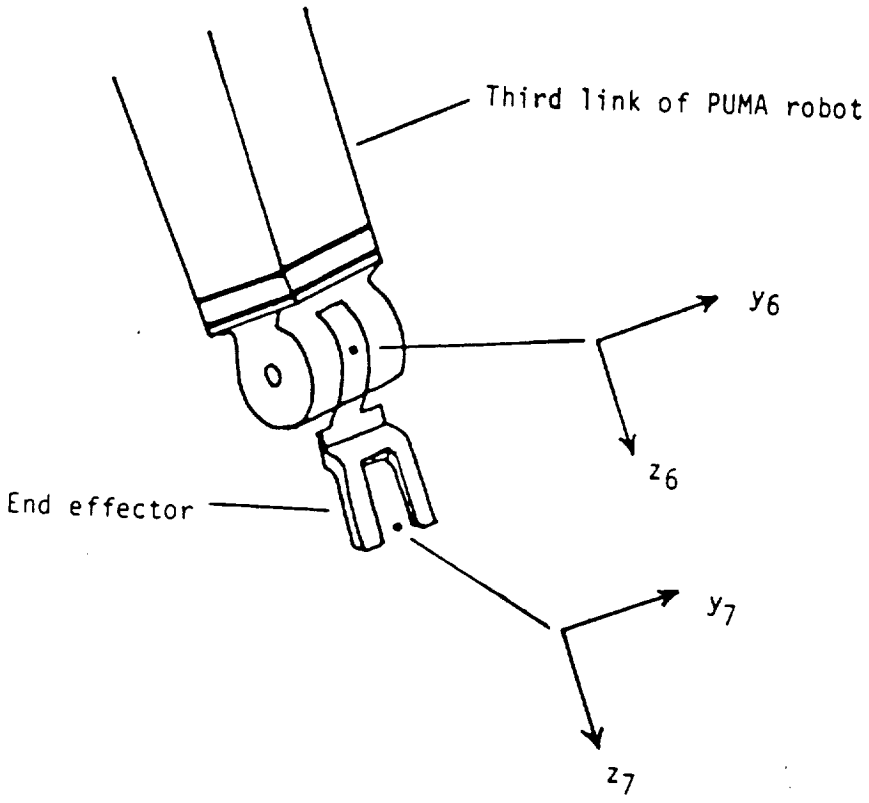

Figure 4. Robot T6 Frame Versus T7 Frame

multiply the ${}^{0}_{6}T$ matrix by a transformation that describes the position and orientation of the tip of the end effector with respect to the ${}^{0}_{6}T$ reference frame.

$$ {}^{0}_{7}T = {}^{0}_{6}T * E \quad \text{where E is also a 4x4 matrix.} \tag{7} $$

Now that we have described how to obtain the position and orientation of the robots end effector given the robots joint angles, the more difficult task is to determine the robot joint angles that will yield a specified position and orientation of the end effector tip. This is referred to as the inverse kinematics calculation. By successively premultiplying both sides of equation by the inverse of the leading term on the left hand side we obtain the following

$$ {}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T\,{}^{5}_{6}T = {}^{0}_{1}T^{-1}({}^{0}_{6}T) \tag{8} $$

$$ {}^{2}_{3}T\,{}^{3}_{4}T\,{}^{4}_{5}T\,{}^{5}_{6}T = {}^{1}_{2}T^{-1}({}^{0}_{1}T^{-1})({}^{0}_{6}T) \tag{9} $$

$$ {}^{3}_{4}T\,{}^{4}_{5}T\,{}^{5}_{6}T = {}^{2}_{3}T^{-1}({}^{1}_{2}T^{-1})({}^{0}_{1}T^{-1})({}^{0}_{6}T) \tag{10} $$

$$ {}^{4}_{5}T\,{}^{5}_{6}T = {}^{3}_{4}T^{-1}({}^{2}_{3}T^{-1})({}^{1}_{2}T^{-1})({}^{0}_{1}T^{-1})({}^{0}_{6}T) \tag{11} $$

$$ {}^{5}_{6}T = {}^{4}_{5}T^{-1}({}^{3}_{4}T^{-1})({}^{2}_{3}T^{-1})({}^{1}_{2}T^{-1})({}^{0}_{1}T^{-1})({}^{0}_{6}T) \tag{12} $$

If we equate terms in equations (8) through (12) the angular values of the six PUMA joints can be found in terms of the Cartesian position and orientation [2]. Again, there is the problem that final solution obtained will be in terms of the components of the ${}^{0}_{6}T$ matrix, whereas what is actually specified is the ${}^{0}_{7}T$ matrix (i.e., the position and orientation of the tip of the end effector). From equation (7) we can obtain the intermediate ${}^{0}_{6}T$ matrix from the actual ${}^{0}_{7}T$ matrix by postmultiplying by the inverse of the end effector transformation so that

$$ {}^{0}_{6}T = {}^{0}_{7}T * E^{-1} \tag{13} $$

5.0 COMMUNICATIONS

While there are two distinct computing nodes, only one node, the local node, allows the user to interface with the system. It is through the local node that the user enters all appropriate data and receives any information from the system. Associated with the local communications processor is a menu. When the two nodes are communicating via the remote and local communication CPUs the user hits the "escape" key on the terminal attached to the local communication processor. This stops communication and enters the user into the index/menu mode. By now typing "m" on the keyboard, the user brings up the menu which has a hierarchical tree structure. Once inside the menu, the operator can call up sub-menus that change such system parameters as operational mode, motion scaling factor, robot end effector length, etc. All data to be shared by processors within a node as well as data to be used by processors within another node are stored in off

board RAM. When the user enters changes from the menu the appropriate data is also changed in off board RAM. Upon exiting the menu the user can re-start the inter node communication by simply hitting the "return" key on the terminal.

Since the local and remote node are located in two different chassis, a way was needed to allow the two nodes to communicate with one another. Another problem was that depending on the operational mode different data would be passed between the nodes. For example in position mode you would want the local node to send a position and orientation matrix, while in joint mode (not implemented yet) you would only want to send six joint angles.

Communication is achieved with two sets of boards, one set resides in each chassis. A communication board set consists of one 32016 CPU board and one BLC-519 I/O board. The BLC-519 has 9 eight bit parallel communication channels and uses the Intel 8255 chip for I/O. Each I/O board has three ports where each port contains three parallel communication channels. Currently, the I/O boards operate in mode 1, which simply means channel A of the port is used for local to remote node communication, channel B of the port is used for remote to local node communication, and the eight lines of channel C are used for hand-shaking.

Within the framework of the C programming language we set a pointer equal to the address of the appropriate channel of the appropriate port of the I/O board. Then by setting the value of whatever the pointer points to we can send a byte of data over the parallel communication link. Likewise, by reading the value of what the pointer points to, we can read what has been sent over the parallel link. Communication is synchronized by the use of a read acknowledge line and a write acknowledge line. If a sender is to send more than one byte of information, it waits for a read acknowledge signal from the receiving side before sending each subsequent byte. The read acknowledge is set by the receiver when its CPU board reads what has been sent to its I/O board. Similarly, the receiver CPU will not read what its I/O board has until it receives a write acknowledge. A write acknowledge is set whenever the sender places a byte of information on its parallel port.

Data sent from the local node to the remote node will always consist of a mode word, local status word, and a remote command word, each being two bytes in length. The bits of the mode word indicate what operational mode the system is currently in, whether indexing is on or not, and whether any parameters have been changed. Similarly the bits of the local status word show the local status, while the bits of the remote command word indicate the functions the remote side is to perform. See Figure 5 for a definition of the bits. Other data to be sent will consist of one or more of the following items FRHC T6 matrix, FRHC Cartesian velocity, FRHC joint angles, robot end effector length or FRHC frame vs. robot frame difference. The last two items will be sent only once after the value of either has been changed by the operator. By deciphering the mode word the local communications CPU determines which data to retrieve from shared RAM and send over the parallel communications link. For example if the mode is joint mode and the parameter change bit is set the local communications CPU will retrieve the mode word, local status word, remote command word, FRHC joint angles, robot end effector length,

- Mode Control Word

Bit	Control Mode
0	Position control
1	Rate control
2	Current control (future use)
3	Joint control (future use)
4	Index mode - On
5	Rotation indexing - On
6 to 14	Unused
15	Parameter change indicator

- FRHC Command Word

Bit	Command
0	Calibration cm'd
1	Servo power on/off cm'd

1c - Robot Command Word

Bit	Command
0	Calibration cm'd
1	Servo power on/off cm'd
2	Unused
3	Gripper open/close cm'd

- Robot Status Word

Bit	Meaning
0	Calibration status
1	Servo power on/off status
2	Joint limit violation in inverse kinematics solution
3	Gripper open or closed status
4	Robot processor (MACS) error
5	Robot UC error
6	Comm processor error
7	Sensor processor error

- FRHC Status Word

Bit	Meaning
0	Calibration status
1	Servo power on/off status
2	Unused for now
3	Unused for now
4	HACS error
5	MIS error
6	Comm processor error
7	Graphics processor error

Figure 5. Mode Status Command Word Bit Definition

and the FRHC vs. robot frame difference. These data items are assembled into a data buffer and sent byte by byte to the remote communications CPU. The first byte sent is a number telling how many bytes of information are contained in the data buffer. Held in the first two bytes of the data buffer is the mode word. Upon deciphering the mode word the receiving node then knows what data is held in the buffer. This data is then deposited in the appropriate places in shared RAM.

The format of the data held in the buffer is the actual binary pattern residing at the memory address that corresponds to the variable you have selected. For unsigned integer variables such as the mode and status word, it is the expected binary representation. However, for non-integer numbers such as the robot end effector or the T6 matrices, the memory location corresponding to a variable represents the non-integer variable as a 64 bit double precision number in the National Semiconductor Series 32000 floating point format. The 64 bit field contains such information as the sign of a number, the value of its exponent and the value of its mantissa. While non-integer information is originally held in a 64 bit field (double precision number) it is first converted to a 32 bit field (single precision number) before placing it in the data buffer. Again, the 32 bit field of the single precision number still contains such information as sign, mantissa, and exponent. The double precision to single precision conversion speeds overall parallel communication at the price of a slight reduction in the accuracy of the numbers transmitted.

In sending the literal contents of a variable's address in memory in floating point format we greatly increase the overall communication throughput, as compared to the option of sending numbers over in ASCII format.

Embedded within the communication software is also a provision for checking the health of each processor. Associated with each processor is a 32 bit long error word. Each bit in the error word corresponds to a specific problem in a specific processor. When a processor detects some problem within itself it then sets the appropriate bit with its error word. Of course these errors must be of a non-catastrophic nature, because if it were to cause a processor to "die" then that processor would be unable to set a bit in its error word. If the remote communications processor finds any of the remote node error words non-zero (i.e., some type of error) it sets the appropriate bit in the remote status word and ships any non-zero error words to the local node. By looking at the bits of the error word the operator can then tell what error has occurred in what processor and take any appropriate action.

6.0 SETPOINT INTERPOLATION

When both the local and remote nodes first became operational it was found that the robot motion, when following position commands generated by the FRHC, was unsatisfactorily jumpy. Upon comparing the cycle rates of the local and remote nodes it was found that the FRHC control CPU was running faster than the PUMA robot control CPU. Since the two processors were running asynchronously this meant that occasionally the robot control processor would miss a position command from the FRHC control CPU.

In an attemptooth out the position commands the robot tries to servo to, a spline fit was made seven if occasionally a point was missed the motion would be smooth and continuouead of smoothing out the FRHC Cartesian commands it was decided to smooth outesults of the robot's inverse kinematics (i.e., a joint space position composed the s six joint values) from an FRHC Cartesian command.

It was deto use a cubic parametric that is both smooth and continuous with the second derv of the curve at the spline knots (the original points we wish splined together) beingrarily set to zero. Referring to Figure 6 where $\underline{p1}$, $\underline{p2}$, and $\underline{p3}$ are the three joint spasition vectors we want splined together and $\underline{T1}$, $\underline{T2}$, and $\underline{T3}$ are the corresponding tnt vectors, then a space curve through $\underline{p2}$ and $\underline{p3}$ is expressed by the equation

$$\underline{p}(u) = A + Bu + Cu^2 + Du^3 \tag{14}$$

where currently $u \leq 1$. From the boundary conditions we get the following relationships:

$$\underline{p}(0) = \underline{p2} \tag{15}$$

$$\underline{p}(1) = \underline{p3} \tag{16}$$

$$\left.\frac{dp}{du}\right|_{u=0} = \underline{T2} \tag{17}$$

$$\left.\frac{dp}{du}\right|_{u=1} = \underline{T3} \tag{18}$$

$$\underline{p}''(0) = \underline{p}''(1) = 0 \tag{19}$$

Using these rationships, we can easily determine that from equation (14)

$$A = \underline{p2} \tag{20}$$

$$B = \underline{T2} \tag{21}$$

$$C = 3 * (\underline{p3} - \underline{p2}) - 2 * \underline{T2} - \underline{T3} \tag{22}$$

$$D = 2 * (\underline{p2} - \underline{p3}) + \underline{T2} + \underline{T3} \tag{23}$$

Further we can determine that

$$T2[i] = \frac{(-6 * (p2[i] - p1[i] + 12 * (p3[i] - p1[i]) - 6 * (p3[i] - p2[i]))}{12} \tag{24}$$

$$T3[i] = \frac{(3 * (p2[i] - p1[i] - 6 * (p3[i] - p1[i]) + 21 * (p3[i] - p2[i]))}{12} \tag{25}$$

where $i = 0, 1, 2, 3, 4, 5$ are the six components of a robot joint space position, $\underline{p1}$ is the third most recent result of the robot inverse kinematics, $\underline{p2}$ is the second most recent result of the robot inverse kinematics and $\underline{p3}$ is the most recent result of the robot inverse kinematics.

Now, using equation (14) we generate seven spline fits that are equally spaced between $u = 0.5$ and $u = 1.5$. So, the spline fit starts midway between $\underline{p3}$ and $\underline{p2}$. It then follows a smooth curve through $\underline{p3}$ and ends at a predicted joint space point halfway past $\underline{p3}$.

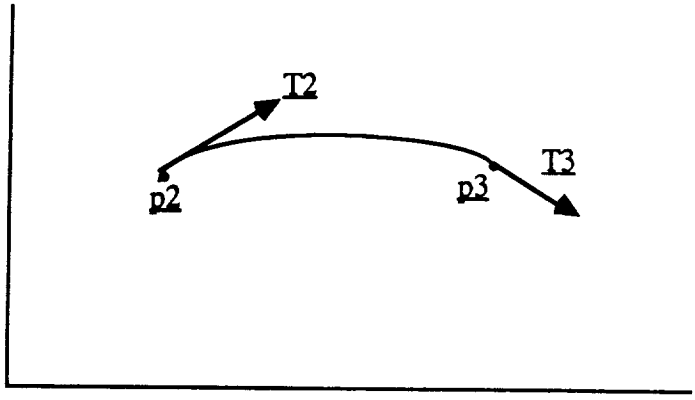

Figure 6. Two dimensional representation of six dimensional space curve

7.0 SYSTEM THROUGHPUT

There are primarily eight tasks that the PUMA robot control CPU performs in a serial fashion.

1) Retrieval of FRHC T6 from shared RAM: This process takes 0.000143 seconds which is equivalent to a rate of 6976.7 Hertz.

2) Workspace adjustment for use in forward kinematics: This process takes 0.001533 seconds which is equivalent to a rate of 6522 Hertz.

3) Forward kinematics: This process takes 0.006 seconds which is equivalent to a rate of 166.6 Hertz.

4) End effector compensation for forward kinematics: This process takes 0.00020 seconds which is equivalent to 5000 Hertz.

5) Placement of PUMA T6 in shared RAM: This process takes 0.0001366 seconds which is equivalent to a rate of 7317 Hertz.

6) Wordjustment for use in inverse kinematics: This process takes 0.00102 seconds equivalent to a rate of 980 Hertz.

7) End compensation for inverse kinematics: This process takes 0.0001925 seconds equivalent to a rate of 5194 Hertz.

8) Invenatics: This process takes 0.01071 seconds which is equivalent to a rate of 93.

Sum these times and including a few other smaller processes that are also part of the rolrol CPU computational workload yields a cycle time of 0.02267 seconds which is at to a rate of 44 Hertz.

Comion time to send a FRHC T6 matrix and receive a PUMA T6 matrix is 0.014 secich is equivalent to a rate of 71.4 Hertz.

The :ontrol CPU cycles at a rate of 25 Hertz in position mode and 27 Hertz in rate mole times of 0.04 seconds and 0.037 seconds respectively). This CPU had a 6 megaick rate.

Sum: of these times in position mode, the minimum amount of time it takes to send and from the FRHC, have the command transmitted to the robot, send the roboti back to the local node and generate position error based force feedback in the FR.0767 seconds, which is equivalent to a rate of 13 Hertz.

All oftware was coded in the high level programming language C, however, a rather inecross compiler (National Semiconductor's GCS) was used.

8.0 CONCNS AND FUTURE WORK

This resents the current framework of our teleoperator system development. Position of a PUMA robot bypassing VAL and using a distributed computing system aptisfactory, albeit with some limitations:

1) SinceMA joint processors' PID control parameters are proprietary information and oe easily accessed and altered, the control flexibility is hence somewhat handi

2) The Ioint rate information, which is crucial for bilateral servo control, is also not a:

3) The cand controller electronics do not provide velocity information, and the velociation by software is not very accurate.

A nevrsal Controller (UC) is under development to replace both of the hand controller ics and the entire PUMA controller in the near future [1]. This UC shall provide eass to the control parameters and easy adaptation of different control

methods. The limitations cited above that appeared in the current teleoperator setup shall be alleviated when the UC is implemented.

The goal of our teleoperator development is to realize a higher throughput bilateral servo control system. The following work is planned to reach this goal:

1) Direct current control, instead of position control, shall be implemented for force or force/position control of the robot arm.

2) Robot dynamics, in addition to kinematics, shall be implemented.

3) Information from robot force torque sensors shall be included in the calculation of hand controller force feedback.

4) To attempt an increase in system throughput, interrupt driving and synchronization of the distributed processor shall be explored.

5) Obtain a more efficient cross compiler.

9.0 REFERENCES

[1] Bejczy, A.K. and Z. Szakaly, "Universal Computer Control System for Space Telerobots," Proceedings of IEEE Conference on Robotics and Automation, Raleigh, NC, March 1987.

[2] Paul, R.P., Robot Manipulators - Mathematics Programming and Control, MIT Press, 1981.

[3] Denavit, J. and R.S. Hartenberg, "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices," Vol. 22, Transactions of the ASME, 19855, pp. 215-221.

[4] Fong, C.P., A.K. Bejczy, and R. Dotson, "Distributed Microcomputer Control System for Advanced Teleoperators," Proceedings of IEEE International Conference on Robotics and Automation, San Francisco, CA, April 7-10, 1986.

[5] Salganicoff, M., E. Austin, and C.P. Fong, "Real-time Simulation of a Distributed Teleoperator System," Proceedings of IASTED (International Association for Science and Technology for Developement) Conference on Applied Simulation and Modeling, Vancouver, BC, June 1986, pp. 594-598.

[6] Pressman, R.S. and J.E. Williams, Numerical Control and Computer Aided Manufacturing, John Wiley and Sons, Inc., 1977.